dsh: A Devil Shell

COMPSCI210 Recitation
14 Sep 2012
Vamsi Thummala
Comments on heap manager

• Q's on pointer manipulation
• Infinite loop
• Space utilization (success rate)
• segfault issues
The fact

Debugging segfaults is hard!

gdb can help

Code walk through is often faster (for this lab)
Shell

- Interactive command interpreter
- A high level language (scripting)
- Interface to the OS
- Provides support for key OS ideas
  - Isolation
  - Concurrency
  - Communication
  - Synchronization
Demo
Unix programming environment

Standard unix programs read a byte stream from standard input (fd==0).
They write their output to standard output (fd==1).

Stdin or stdout might be bound to a file, pipe, device, or network socket.

If the parent sets it up, the program doesn’t even have to know.
That style makes it easy to combine simple programs using pipes or files.
Shell Concepts

- Process creation
- Execution
- Input/Output redirection
- Pipelines
- Job control
  - Process groups
  - Sessions
  - Foreground/background jobs
    - Given that many processes can be executed concurrently, which processes should have accesses to the keyboard/screen (I/O)?
  - Signals
    - SIGSEGV (segfault), SIGINT, SIGCONT
Unix fork/exec/exit/wait syscalls

int pid = fork();
Create a new process that is a clone of its parent.

exec("program", [argvp, envp]);
Overlay the calling process with a new program, and transfer control to it.

exit(status);
Exit with status, destroying the process.
Note: this is not the only way for a process to exit!

int pid = wait(&status);
Wait for exit (or other status change) of a child, and “reap” its exit status.
Note: child may have exited before parent calls wait!
Process creation and execution

while (1) {
    printf("$");
    command = readnparse(args);
    switch (pid = fork()) { // new process; concurrency
        case -1:
            perror("Failed to fork\n");
        case 0: // child when pid = 0
            exec (command, args, 0); // run command
        default: // parent pid > 0
            waitpid(pid, NULL, 0); // wait until child is done
    }
}
Input/Output (I/O)

- I/O through file descriptors
  - File descriptor may be for a file, terminal, ...
- Example calls
  - read(fd, buf, sizeof(buf));
  - write(fd, buf, sizeof(buf));
- Convention:
  - 0: input
  - 1: output
  - 2: error
- Child inherits open file descriptors from parents
I/O redirection (\(<\) \(>\))

• Example: “ls > tmpFile”
• Modify \(dsh\) to insert before exec:
  
  ```
  close(1); // release fd 1
  fd = create(“tmpFile”, 0666); // fd will be 1
  ```

• No modifications to “ls”!
• “ls” could be writing to file, terminal, etc., but programmer of “ls” doesn’t need to know
Pipeline: Chaining processes

- One-way communication channel
- Symbol: |

```c
int fdarray[2]; char buffer[100];
pipe(fdarray);
write(fdarray[1], "hello world", 11);
read(fdarray[0], buffer, sizeof(buffer));
```
int fdarray[2];
char buffer[100];
pipe(fdarray);
pipe(fdarray);
switch (pid = fork()) {
    case -1: perror("fork failed");
    case 0: write(fdarray[1], "hello world", 5);
    default: n = read(fdarray[0], buffer, sizeof(buffer)); //block until data is available
}

How does the pipes in shell, i.e, “ls | wc”? 

dup2(newfd, oldfd); // duplicates fd; closes and copies at one shot
Process groups

– A process group is a collection of (related) processes. Each group has a process group ID.
– Each group has a group leader who pid = pgid
– To get the group ID of a process:
  \[\text{pid}_t \text{ getpgrp}(\text{void})\]
– A process may join an existing group, create a new group.
  \[\text{int setpgid}(\text{pid}_t, \text{pid}, \text{pid}_t, \text{pgid})\]
– A signal can be sent to the whole group of processes.
pid_t spawn_job(bool fg, pid_t pgrp) {

    int ctty = -1;
    pid_t pid;

    switch (pid = fork()) {
    case -1: /* fork failure */
        return pid;
    case 0: /* child */
        /* establish a new process group, and put the child in
        * foreground if requested
        * Q: what if setpgid fails?
        */
        if (pgrp < 0)
            pgrp = getpid();

            if (setpgid(0,pgrp) == 0 && fg) // If success and fg is set
                tcsetpgrp(ctty, pgrp); // assign the terminal

            return 0;
    default: /* parent */
        /* establish child process group here too. */
        if (pgrp < 0)
            pgrp = pid;
    
            if (setpgid(pid, pgrp))
                return pid;
            
            setpgid(pid, pgrp);

            return pid;
    }
}